

# Software-Projekt 2 2013/14

VAK 03-BA-901.02

## Architekturbeschreibung

IT\_R3V0LUT10N

Sebastian Bredehöft	sbrede@tzi.de	2751589
Patrick Damrow	damsen@tzi.de	2056170
Tobias Dellert	tode@tzi.de	2936941
Tim Ellhoff	tellhoff@tzi.de	2520913
Daniel Pupat	dpupat@tzi.de	2703053

## Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>4</b>
1.1	Zweck . . . . .	4
1.2	Status . . . . .	4
1.3	Definitionen, Akronyme und Abkürzungen . . . . .	4
1.4	Referenzen . . . . .	4
1.5	Übersicht über das Dokument . . . . .	4
<b>2</b>	<b>Globale Analyse</b>	<b>4</b>
2.1	Einflussfaktoren . . . . .	4
2.1.1	Organisatorische Faktoren . . . . .	5
2.1.2	Technische Faktoren . . . . .	6
2.1.3	Produkt Faktoren . . . . .	8
2.2	Probleme und Strategien . . . . .	9
<b>3</b>	<b>Konzeptionelle Sicht</b>	<b>18</b>
3.1	Überblick . . . . .	18
3.2	Serverkomponente . . . . .	20
3.3	Clientkomponente . . . . .	21
<b>4</b>	<b>Modulsicht</b>	<b>22</b>
4.1	Pakete . . . . .	22
4.1.1	Paket bibclient . . . . .	23
4.1.2	Pakete bibcommon . . . . .	24
4.1.3	Paket bibjsf . . . . .	25
<b>5</b>	<b>Datensicht</b>	<b>27</b>
<b>6</b>	<b>Ausführungssicht</b>	<b>27</b>
<b>7</b>	<b>Zusammenhänge zwischen Anwendungsfällen und Architektur</b>	<b>28</b>
<b>8</b>	<b>Evolution</b>	<b>29</b>

## Abbildungsverzeichnis

iiiiiii HEAD

1	Konzeptionelle Sicht (Klein)	18
2	Konzeptionelle Sicht	19
3	Konzeptionelle Sicht Server	20
4	Konzeptionelle Sicht Client	21
5	Pakete Übersicht	22
6	Paket bibclient	23
7	Paket bibcommon	24
8	Paketübersicht bibjsf	25
9	Paket bibjsf	26
10	Ausführungssicht	28

=====

1	Konzeptionelle Sicht (Klein)	19
2	Konzeptionelle Sicht	20
3	Konzeptionelle Sicht Server	21
4	Konzeptionelle Sicht Client	22
5	Pakete Übersicht	24
6	Paket bibclient	25
7	Paket bibcommon	26
8	Paket bibclient	27
9	Ausführungssicht	29

LLLLLLL 3aec9c381cd8e3a5835c4fe21460557c4c9046f

## Tabellenverzeichnis

1	Organisatorische Faktoren	5
2	Technische Faktoren	6
3	Produkt Faktoren	8

## Version und Änderungsgeschichte

*Die aktuelle Versionsnummer des Dokumentes sollte eindeutig und gut zu identifizieren sein, hier und optimalerweise auf dem Titelblatt.*

Version	Datum	Änderungen
1.0	25.11.2013	Dokumentvorlage als initiale Fassung kopiert
1.1	08.12.2013	Einflussfaktoren

## 1 Einführung

### 1.1 Zweck

*Was ist der Zweck dieser Architekturbeschreibung? Wer sind die LeserInnen?*

### 1.2 Status

### 1.3 Definitionen, Akronyme und Abkürzungen

### 1.4 Referenzen

### 1.5 Übersicht über das Dokument

## 2 Globale Analyse

### 2.1 Einflussfaktoren

Die Einflussfaktoren werden im Folgenden unterteilt in:

- Organisatorische Faktoren
- Technische Faktoren
- Produktfaktoren

### 2.1.1 Organisatorische Faktoren

Tabelle 1: Organisatorische Faktoren

<b>O1</b>	<b>Time-To-Market</b>
<b>O2</b>	<b>Auslieferung von Produktfunktionen</b>
<b>O3</b>	<b>Budget</b>
<b>O4</b>	<b>Kenntnisse in Java und Android</b>
<b>O5</b>	<b>Kenntnisse in J-Unit</b>
<b>O6</b>	<b>Anzahl der Entwickler</b>

<b>O1</b>	<b>Time-To-Market</b>
Faktor	Auslieferungsdatum 23.02.2014
Flexibilität und Veränderlichkeit	Die Deadline kann nicht verändert werden
Auswirkungen	Die Software muss zum Abgabedatum lauffähig sein

<b>O2</b>	<b>Auslieferung von Produktfunktionen</b>
Faktor	Alle Mindestanforderungen
Flexibilität und Veränderlichkeit	Es müssen alle Mindestanforderungen erfüllt sein, sind jedoch vom Kunden oder beim verlassen eines Gruppenmitglieds veränderbar
Auswirkungen	Architektur muss alle Mindestanforderungen abdecken, es muss darauf geachtet werden, dass diese sich im Verlauf noch ändern

<b>O3</b>	<b>Budget</b>
Faktor	Kein finanzielles Budget
Flexibilität und Veränderlichkeit	Es werden keine finanziellen Unterstützungen für das Produkt geben
Auswirkungen	Es können keine Kostenpflichtigen Dienste in Anspruch genommen werden

<b>O4</b>	<b>Kenntnisse in Java und Android</b>
Faktor	Kenntnisse der Entwickler in Java und Android
Flexibilität und Veränderlichkeit	Kenntnisse sind nicht flexibel, es muss in Java programmiert werden und über Smartphone laufen. Die Kenntnisse können sich im Laufe ändern, durch neue Erfahrungen und neu erworbenen Kenntnissen
Auswirkungen	Bei wenig Kenntnissen muss mehr Zeit eingeplant werden um sich diese anzueignen

<b>O5</b>	<b>Kenntnisse in J-Unit</b>
Faktor	Kenntnisse in J-Unit Tests
Flexibilität und Veränderlichkeit	Da Tests mit J-Unit gefordert werden, sind diese nicht verhandelbar oder Flexibel
Auswirkungen	Bei unzureichenden Tests kann es später beim Programm zu Problemen kommen, da Fehler spät oder gar nicht erkannt werden

<b>O6</b>	<b>Anzahl der Entwickler</b>
Faktor	Die Anzahl der Entwickler
Flexibilität und Veränderlichkeit	Es können keine neuen Gruppenmitglieder dazukommen, es können aber jederzeit Gruppenmitglieder wegfallen
Auswirkungen	Wenn Gruppenmitglieder wegfallen, müssen die restlichen mehr Arbeiten und mehr Zeit einplanen. Auch müssen Projektplan und Architektur neu angepasst werden.

### 2.1.2 Technische Faktoren

Tabelle 2: Technische Faktoren

<b>T1</b>	<b>Software funktioniert unter Windows, Linux und MacOS</b>
<b>T2</b>	<b>Software funktioniert als App(Android 2.3 oder höher)</b>
<b>T3</b>	<b>SQL-Datenbank</b>
<b>T4</b>	<b>Mehrere parallele Nutzer</b>
<b>T5</b>	<b>Client-Server System</b>
<b>T6</b>	<b>Benutzerschnittstelle</b>
<b>T7</b>	<b>Implementierungssprache Java</b>
<b>T8</b>	<b>Beschränkungsfreiheit für Fremdbibliotheken</b>

<b>T1</b>	<b>Software funktioniert unter Windows, Linux und MacOS</b>
Faktor	Die Software muss auf den Betriebssystemen Windows, Linux und MacOS laufen
Flexibilität und Veränderlichkeit	nicht Flexibel, da dies zu den Mindestanforderungen gehört. Veränderungen können jederzeit vom Kunden vorgenommen werden.
Auswirkungen	Die Entwickler müssen sich mit allen Betriebsprogrammen befassen und sichergehen, dass es auf allen funktioniert

<b>T2</b>	<b>Software funktioniert als App (Android 2.3 oder höher)</b>
Faktor	Die Software muss als Android App auf einem Smartphone laufen
Flexibilität und Veränderlichkeit	nicht Flexibel, da dies zu den Mindestanforderungen gehört. Veränderungen können jederzeit vom Kunden vorgenommen werden.
Auswirkungen	Die Software muss wie gefordert als App auf einem Android-Smartphone laufen

<b>T3</b>	<b>SQL-Datenbank</b>
Faktor	Software läuft über eine relationale Datenbank
Flexibilität und Veränderlichkeit	Flexibel jedoch muss eine Datenbank mit SQL oder SQL-ähnlichen abfragen verwendet werden
Auswirkungen	Es muss eine relationale Datenbank für die serverseitige Persistenz benutzt werden. Es muss eine Datenbank mit SQL oder SQL-ähnlichen abfragen verwendet werden

<b>T4</b>	<b>Mehrere parallele Nutzer</b>
Faktor	Es greifen mehrere Nutzer zur gleichen Zeit auf die Software zu
Flexibilität und Veränderlichkeit	Es ist uns überlassen, wie viele Nutzer zur gleichen Zeit auf das System zugreifen dürfen
Auswirkungen	Die Software muss darauf ausgelegt sein, mehrere Nutzer zur gleichen Zeit zu verwalten

<b>T5</b>	<b>Client-Server System</b>
Faktor	Die Software arbeitet über ein Client-Server System
Flexibilität und Veränderlichkeit	Da wir übers Internet auf den Server zugreifen müssen, ist es notwendig ein Server-Client System zu verwenden
Auswirkungen	Die Implementierung wird in Server und Client aufgeteilt(siehe 3) Übers Internet werden die Daten zwischen Server und Client ausgetauscht

<b>T6</b>	<b>Benutzerschnittstelle</b>
Faktor	Es sollte eine übersichtliche und ansprechende GUI geben
Flexibilität und Veränderlichkeit	Die Gestaltung der GUI ist uns überlassen
Auswirkungen	Für eine Benutzerfreundliche Gestaltung sind gute Kenntnisse in XHTML notwendig

<b>T7</b>	<b>Implementierungssprache Java</b>
Faktor	Die Software muss in Java 5 oder höher geschrieben werden
Flexibilität und Veränderlichkeit	Nicht Flexibel, da dies zu den Mindestanforderungen gehört
Auswirkungen	Die Software muss in Java geschrieben werden, daher müssen alle Entwickler diese Sprache beherrschen

<b>T8</b>	<b>Beschränkungsfreiheit für Fremdbibliotheken</b>
Faktor	Fremdbibliotheken dürfen für den Einsatz in Forschung und Lehre keine Beschränkungen aufweisen
Flexibilität und Veränderlichkeit	Nicht Flexibel, da dies zu den Mindestanforderungen gehört
Auswirkungen	Es darf keine Software oder Bibliothek verwendet werden, die Kostenpflichtig ist

### 2.1.3 Produkt Faktoren

Tabelle 3: Produkt Faktoren

<b>P1</b>	<b>Mindestanforderung</b>
<b>P2</b>	<b>Performanz</b>
<b>P3</b>	<b>Benutzerrechte</b>
<b>P4</b>	<b>Fehlererkennung</b>

<b>P1</b>	<b>Mindestanforderung</b>
Faktor	Das Produkt muss alle Mindestanforderungen enthalten
Flexibilität und Veränderlichkeit	Alle Anforderungen müssen zum Bestehen erfüllt werden. Die Anforderungen können vom Kunden oder Dozenten verändert werden oder die Anforderungen werden bei einem Austritt eines Mitglieds verringert.
Auswirkungen	Es müssen alle Mindestanforderungen implementiert werden

<b>P2</b>	<b>Performanz</b>
Faktor	Möglichst schnelle Ausführungszeiten
Flexibilität und Veränderlichkeit	Flexibel, da nichts davon in den Mindestanforderungen steht
Auswirkungen	Es sollte bei der Implementierung auf einen schnellen Datenaustausch zwischen Server und Client geachtet werden



<b>P3</b>	<b>Benutzerrechte</b>
Faktor	Es gibt verschiedene Benutzer mit unterschiedlichen Rechten
Flexibilität und Veränderlichkeit	Nicht Flexibel, da dies vom Kunden gefordert wird
Auswirkungen	Es müssen unterschiedliche Benutzer implementiert werden, die unterschiedliche Rechte haben und diese auch nicht überschreiten dürfen

<b>P4</b>	<b>Fehlererkennung</b>
Faktor	Fehler sollten von der Software erkannt werden und entsprechend behandelt werden
Flexibilität und Veränderlichkeit	Flexibel, da dies nicht ausdrücklich vom Kunden gefordert wird
Auswirkungen	Fehler müssen erkannt werden und durch Exception muss es dann entsprechend Korrigiert werden. Die Software sollte weiter laufen

## 2.2 Probleme und Strategien

Folgenden Probleme haben wir identifiziert:

Nummer	Faktoren
1	Zeitprobleme
2	Mangelnde Kenntnisse in Java
3	Mangelnde Kenntnisse in Android
4	Mangelnde Kenntnisse von Datenbanksystemen
5	Unzureichende Softwaretests
6	Ausfall eines Gruppenmitglieds
7	Mehrere parallele Nutzer
8	Performanz
9	unterschiedliche Benutzerrechte

Diese versuchen wir mit folgenden Strategien zu überbrücken:

**1 Zeitprobleme**

Es gibt einen festgesetzten Abgabetermin, der eingehalten werden muss

**Einflussfaktoren**

- O1 Time-To-Market
- O2 Auslieferung von Produktfunktionen
- O4 Kenntnisse in Java und Android
- O5 Kenntnisse in J-Unit
- O6 Anzahl der Entwickler
- P1 Mindestanforderungen

**Lösung**

- Strategie 1: Modularisierung für paralleles Arbeiten  
Durch Modularisierung können mehrere Entwickler zur gleichen Zeit am Projekt arbeiten und die Module unabhängig voneinander implementieren. Diese werden dann später zusammengesetzt
- Strategie 2: Bibliotheken Benutzen  
Es werden bereits vorhandene Java Bibliotheken verwendet, dies spart Zeit, da man dann nicht alles neu schreiben muss.

Es werden beide Strategien verwendet.

## 2 Mangelnde Kenntnisse in Java

Es werden Vorkenntnisse in Java vorausgesetzt, ohne diese könnte es zu großen Problemen kommen, da ohne ausreichende Kenntnisse das Programm nicht realisiert werden kann

### **Einflussfaktoren**

- O1 Time-To-Market
- O2 Auslieferung von Produktfunktionen
- O4 Kenntnisse in Java und Android
- O6 Anzahl der Entwickler
- T1 Software funktioniert unter Windows, Linux und MacOS
- T5 Client-Server System
- T7 Implementierungssprache Java
- P1 Mindestanforderungen

### **Lösung**

- Strategie 1: Modularisierung  
Der Code wird in verschiedene Module aufgeteilt, wenn ein Gruppenmitglied nicht genügend Kenntnisse besitzt, kann dieses Modul von einem anderen Mitglied neu erstellt werden und der inkompetente Entwickler kann keinen Schaden auf andere Module auswirken.
- Strategie 2: Aufteilen in Server und Client  
Die Implementierung wird unter den Entwicklern so aufgeteilt, dass ein Teil den Client und ein Teil den Server macht, so müssen sich die Gruppenmitglieder nicht Kenntnisse in beiden Teilen aneignen.

Es werden beide Strategien verwendet.

### 3 Mangelnde Kenntnisse in Android

Es werden Kenntnisse in Android vorausgesetzt, da eine App entwickelt werden muss.

#### **Einflussfaktoren**

- O1 Time-To-Market
- O2 Auslieferung von Produktfunktionen
- O4 Kenntnisse in Java und Android
- O6 Anzahl der Entwickler
- T2 Software funktioniert als App(Android 2.3 oder höher)
- T5 Client-Server System
- T6 Benutzerschnittstelle
- T7 Implementierungssprache Java
- P1 Mindestanforderungen

#### **Lösung**

- Strategie 1: Modularisierung  
Der Code wird in verschiedene Module aufgeteilt, wenn ein Gruppenmitglied nicht genügend Kenntnisse besitzt, kann dieses Modul von einem anderen Mitglied neu erstellt werden und der inkompetente Entwickler kann keinen Schaden auf andere Module auswirken.
- Strategie 2: Bearbeitung von Gruppenmitgliedern mit Android-Erfahrung  
Wir werden die Implementierung einem Gruppenmitglied überlassen, welches bereits Erfahrung mit Android hat. So müssen sich die anderen nicht in Android einarbeiten und können sich bei Fragen an diesen wenden.

Es werden beide Strategien verfolgt, sollte das Gruppenmitglied mit Android zeitlich oder fachlich nicht klarkommen, wird ein weiteres Gruppenmitglied sich mit Android beschäftigen.

#### 4 Mangelnde Kenntnisse in Datenbanksystemen

Es werden Kenntnisse in Datenbanksystemen vorausgesetzt, da wir für die Bibliothek eine Datenbank verwenden. Dabei werden SQL- oder SQL-ähnliche Abfragen verwendet und entsprechende Kenntnisse verlangt

##### **Einflussfaktoren**

- O1 Time-To-Market
- O2 Auslieferung von Produktfunktionen
- O7 Kenntnisse in Datenbanksystemen
- O6 Anzahl der Entwickler
- T5 Client-Server System
- T7 Implementierungssprache Java

##### **Lösung**

- Strategie 1: Bearbeitung von Gruppenmitgliedern mit Erfahrung in Datenbanksystemen  
Wir werden die Implementierung Gruppenmitgliedern überlassen, welches bereits Erfahrung mit Datenbanksystemen hat. So müssen sich die anderen nicht in Datenbanksystemen einarbeiten und können sich bei Fragen an diesen wenden.

**5 Unzureichende Softwaretests**

Es werden genügend Tests benötigt, welche Module und Komponenten testen, ob diese funktionieren

**Einflussfaktoren**

- O1 Time-To-Market
- O5 Kenntnisse in J-Unit
- T7 Implementierungssprache Java
- P1 Mindestanforderungen
- P2 Performanz
- P3 Benutzerrechte
- P4 Fehlererkennung

**Lösung**

- Strategie 1: Modularisierung  
Es werden Tests für die jeweilig implementierten Module geschrieben, ob diese ihren Zweck erfüllen und danach werden Module zusammen getestet.

6 Ausfall eines Gruppenmitglieds

Es kann jederzeit ein Gruppenmitglied aus der Gruppe austreten oder durch Krankheit etc. für eine gewisse Zeit ausfallen.

**Einflussfaktoren**

- O1 Time-To-Market
- O2 Auslieferung von Produktfunktionen
- O6 Anzahl der Entwickler
- P1 Mindestanforderungen

**Lösung**

- Strategie 1: Modularisierung  
Der Code wird in verschiedene Module aufgeteilt, welche von einem Entwickler bearbeitet werden. Wenn nun ein Entwickler ausfällt, kann ein Modul von einem anderen Entwickler übernommen werden.

7 Mehrere parallele Nutzer

Es greifen mehrere Nutzer zur gleichen Zeit auf das System zu, auf welche der Server antworten muss. Dabei soll der Server die Daten nicht an alle Clients senden

**Einflussfaktoren**

- O1 Time-To-Market
- O3 Budget
- T3 SQL-Datenbank
- T4 Mehrere parallele Nutzer
- P1 Mindestanforderungen
- P2 Performanz
- P3 Benutzerrechte

**Lösung**

- Strategie 1: Thread  
Die Clients bekommen jeweils einen Thread, somit können sie zeitgleich auf den Server zugreifen und bekommen nur ihre Daten zurück.



8 Performanz

Die Software sollte kurze Ausführungszeiten haben. Dabei ist zu beachten, dass die Software/App auch auf Geräten mit geringer Leistung schnell und problemlos läuft

**Einflussfaktoren**

- O1 Time-To-Market
- T1 Software funktioniert unter Windows, Linux und MacOS
- T2 Software funktioniert als App(Android 2.3 oder höher)
- T3 SQL-Datenbank
- T4 Mehrere parallele Nutzer
- P1 Mindestanforderungen
- P2 Performanz
- P3 Benutzerrechte

**Lösung**

- Strategie 1: Code effizient schreiben  
Den Code effizient schreiben, damit die Software kurze Ausführungszeiten hat.

9 unterschiedliche Benutzerrechte

Die Software hat unterschiedliche Benutzer, welche unterschiedliche Rechte besitzen und diese müssen unterschieden werden

### **Einflussfaktoren**

- O1 Time-To-Market
- T3 SQL-Datenbank
- T4 Mehrere parallele Nutzer
- P1 Mindestanforderungen
- P3 Benutzerrechte

### **Lösung**

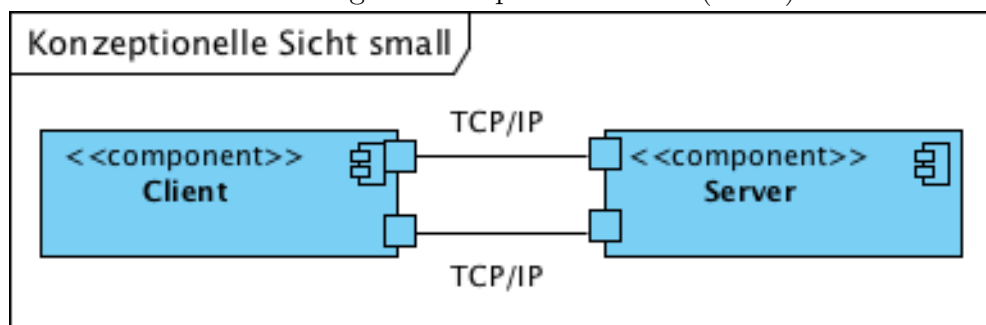
- Strategie 1: Identifikation durch Group Id  
In der Datenbank wird eine Group Id eingefügt, welche dann die verschiedenen Nutzer speichert. Über diese werden dann die verschiedenen Rechte geregelt.

## 3 Konzeptionelle Sicht

Wir haben mithilfe von UML-Diagrammen die konzeptionelle Sicht realisiert. Im Folgenden werden die einzelnen Diagramme aufgezeigt und beschrieben und in nachfolgenden Sichten zusätzlich verfeinert und konkretisiert.

### 3.1 Überblick

Abbildung 1: Konzeptionelle Sicht (Klein)

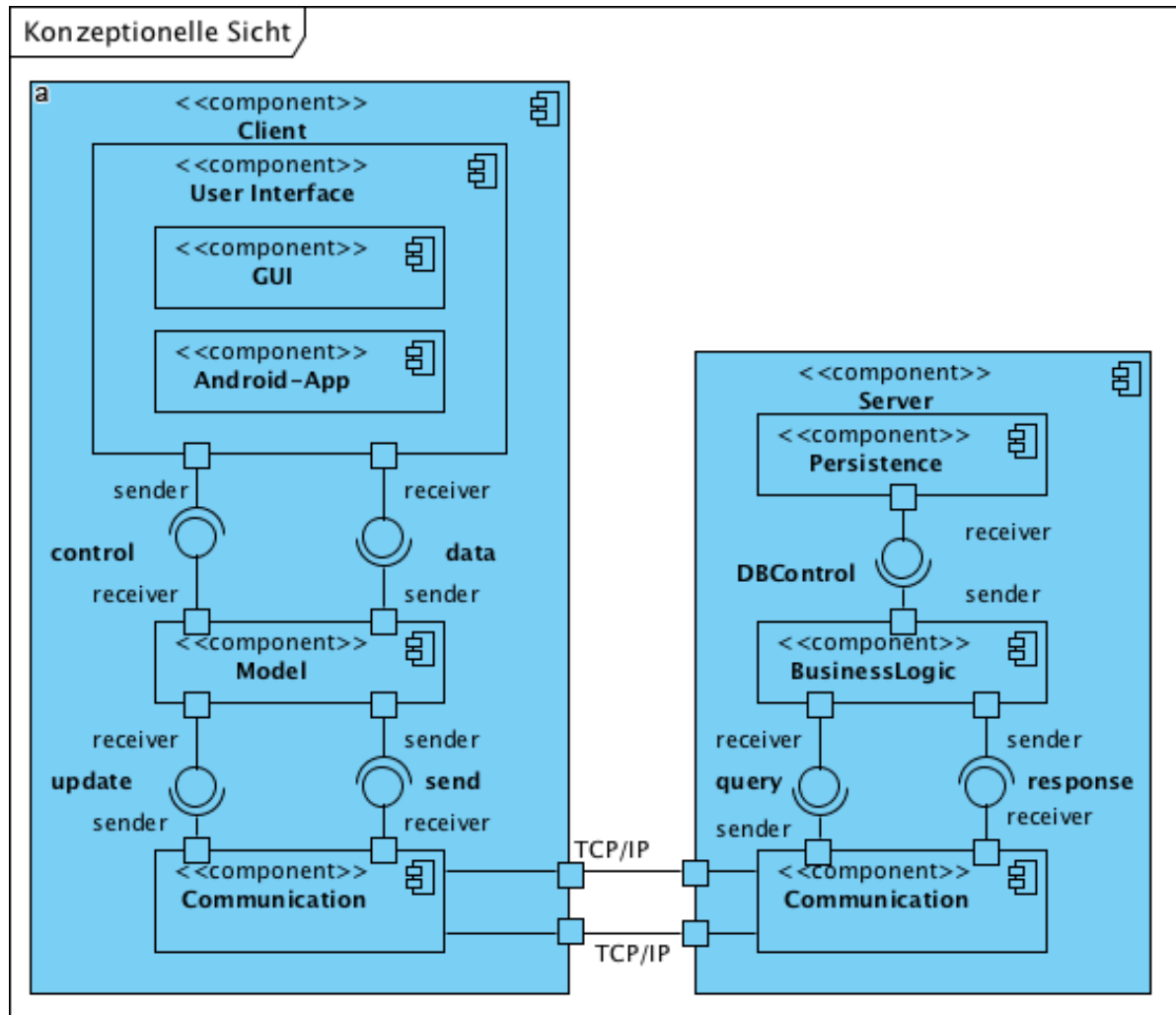


Unsere Architektur besteht aus zwei grundlegenden Komponenten, der Serverkomponente und der Clientkomponente. Diese Komponenten beinhalten wiederum weitere Komponenten. Auf der einen Seite haben wir unsere Serverkomponente, die alle benötigten Daten der Medien, der Nutzer und der Ausleihvorgänge der Bibliothek speichert.

Auf der anderen Seite, der Clientkomponente, muss zwischen zwei Teilen unterschieden werden. Einmal der Gui-Client, welcher sich in erster Linie an die Bibliothekare richtet und der mobile Android-Client, der sich ausschließlich an die Leser richtet.

Der Gui-Client stellt für die Bibliothekare alle benötigten Funktionen bereit um eine Bibliothek zu verwalten. Der Android-Client ermöglicht dem Leser sich Mediendetails, Ausleihstatus, seine eigene Ausleihhistorie sowie persönliche Daten anzeigen zu lassen. Desweiteren kann der Leser sich mittels der App Bücher zur Ausleihe vormerken und Informationen über die Bibliothek abrufen.

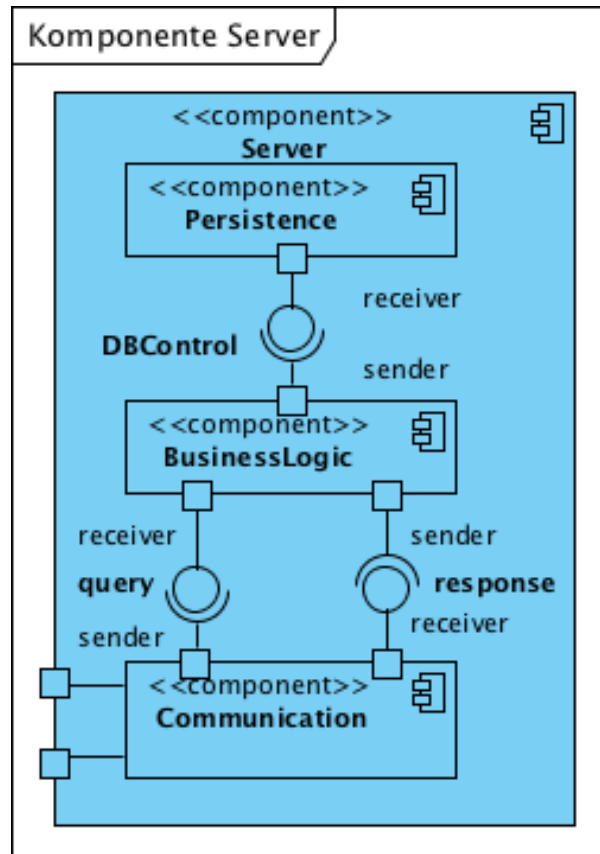
Abbildung 2: Konzeptionelle Sicht



Als Architekturstil verwenden wir das Model-View-Controller-Pattern.

## 3.2 Serverkomponente

Abbildung 3: Konzeptionelle Sicht Server



Die Serverkomponente besteht aus insgesamt drei Teilkomponenten, welche sich wie folgt aufgliedern:

- Communication

Die Komponente Communication nimmt Anfragen des Clients entgegen und leitet sie an die Komponente BusinessLogic weiter wo die Anfragen verarbeitet werden und sendet die Ergebnisse zurück an den Client.

- BusinessLogic

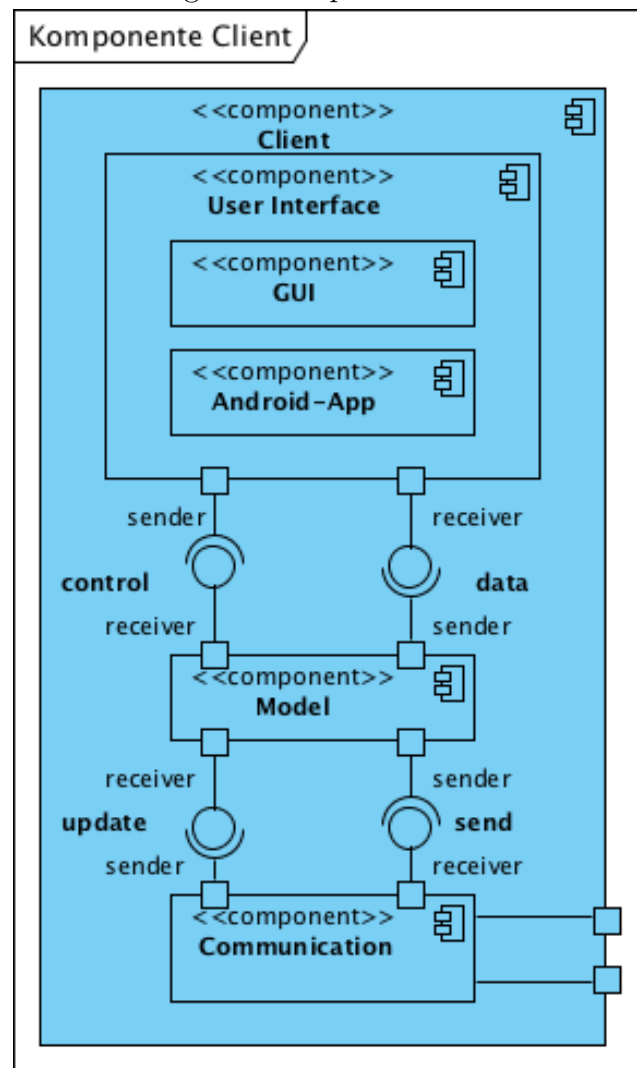
Die Komponente BusinessLogic dient zum verarbeiten der Anfragen und leitet diese verarbeiteten Anfragen dann an die Komponente Persistence weiter.

- Persistence

Die Komponente Persistence ist die Schnittstelle zur Datenbank. Über das Interface DBControl werden die verarbeiteten Anfragen von der Komponente BusinessLogic empfangen und in Datenbankabfragen umgewandelt, welche dann von der Datenbank entgegen genommen werden.

### 3.3 Clientkomponente

Abbildung 4: Konzeptionelle Sicht Client



Die Clientkomponente besteht so wie die Serverkomponente aus drei Teilkomponenten, welche sich wie folgt aufgliedern:

- Communication

Die Komponente Communication sendet Anfragen des Clients an den Server, welche dort verarbeitet werden und nimmt die Ergebnisse entgegen um diese an die Komponente Model zu übergeben wo die Ergebnisse der Anfrage weiter verarbeitet werden.

- Model

Die Komponente Model nimmt Ergebnisse von der Komponente Communication

entgegen und schickt diese an die Komponente User Interface.

- User Interface

Die Komponente User Interface muss in zwei unterschiedliche Komponenten zerlegt werden:

- GUI

Die GUI richtet sich in erster Linie an Bibliothekare.

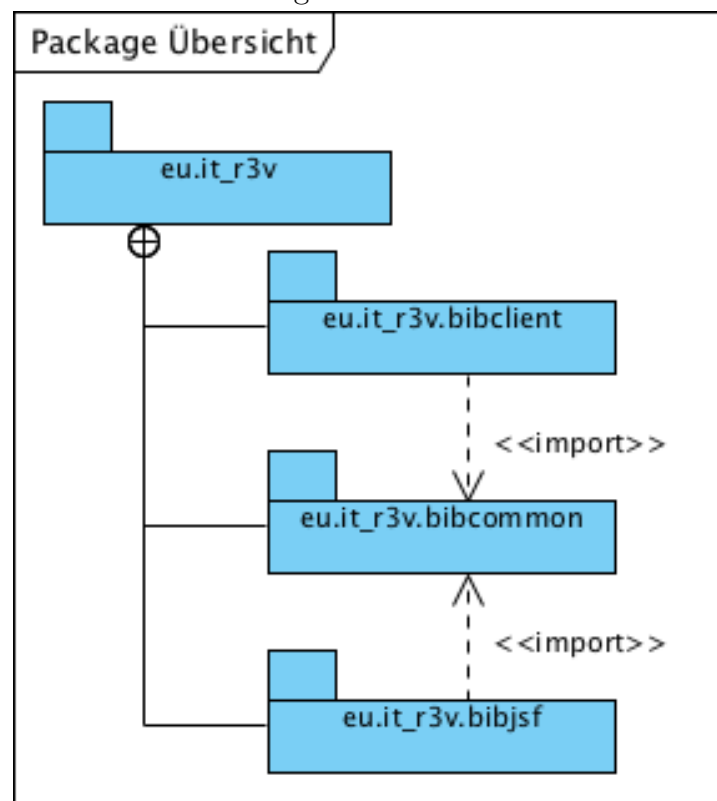
- Android-App

## 4 Modulsicht

### 4.1 Pakete

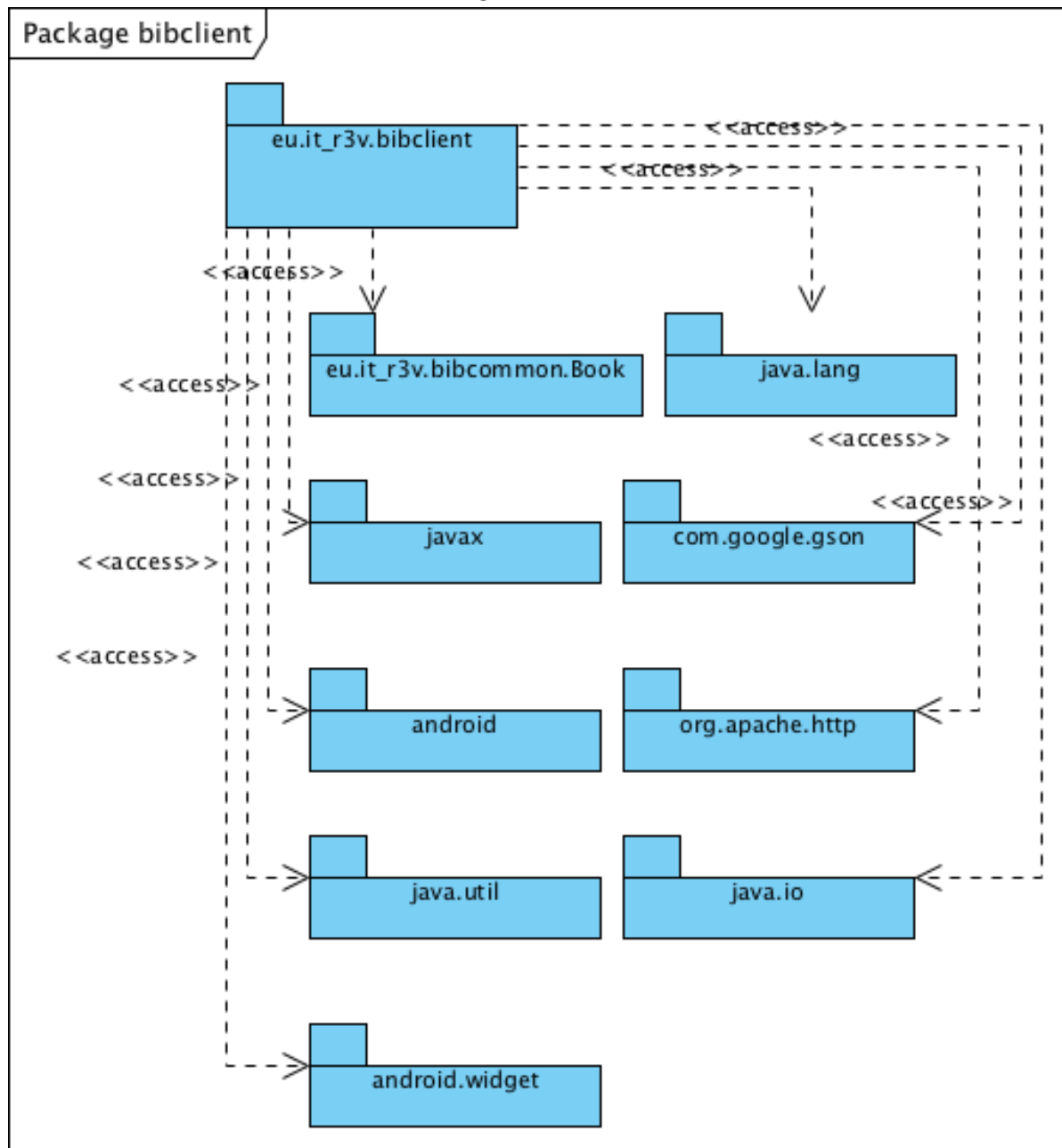
Wir haben ein Hauptpaket `eu.it_r3v` in dem sich weitere Unterpakete befinden. Diese dienen der Bündelung gemeinsamer Quellcodedateien.

Abbildung 5: Pakete Übersicht



#### 4.1.1 Paket bibclient

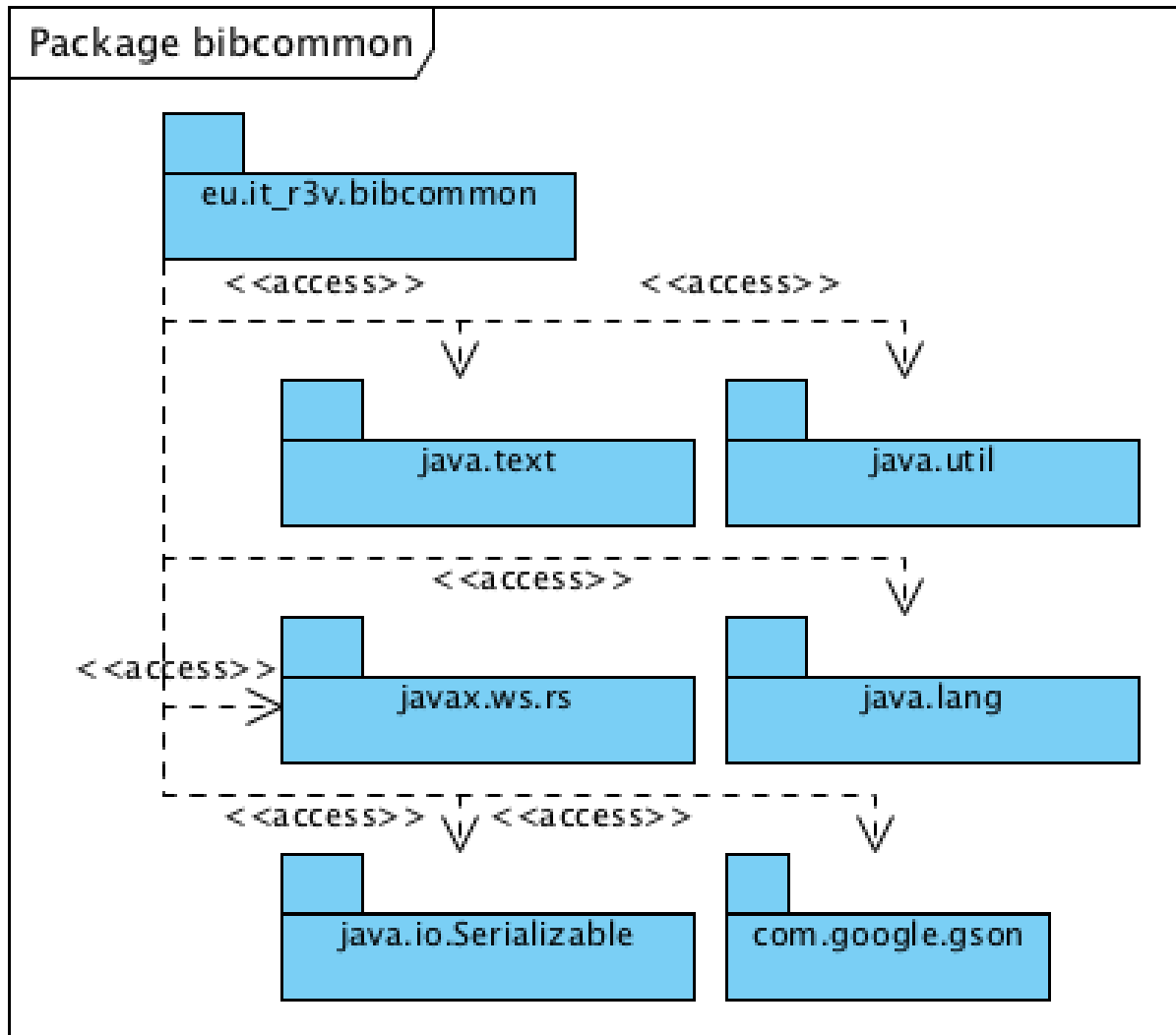
Abbildung 6: Paket bibclient





#### 4.1.2 Pakete bibcommon

Abbildung 7: Paket bibcommon



### 4.1.3 Paket bibjsf

Abbildung 8: Paketübersicht bibjsf

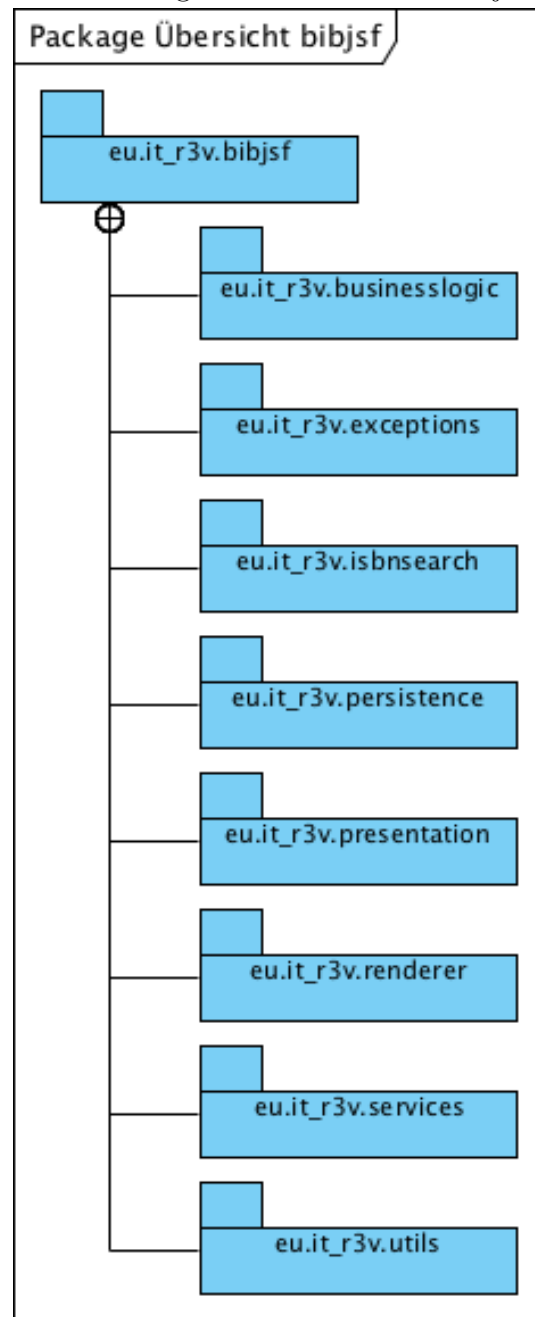
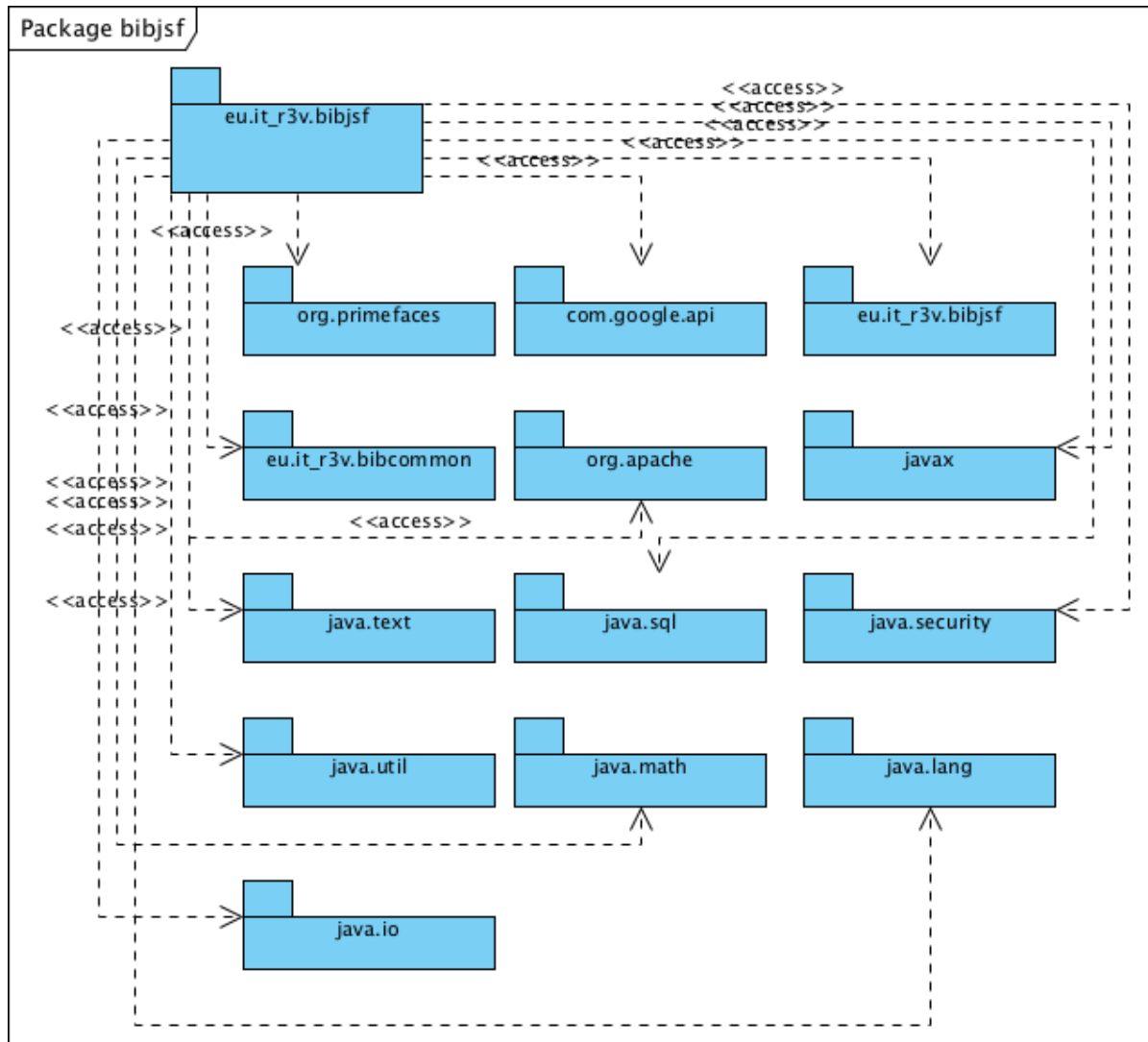


Abbildung 9: Paket bibjsf



Diese Sicht beschreibt den statischen Aufbau des Systems mit Hilfe von Modulen, Subsystemen, Schichten und Schnittstellen. Diese Sicht ist hierarchisch, d.h. Module werden in Teilmodule zerlegt. Die Zerlegung endet bei Modulen, die ein klar umrissenes Arbeitspaket für eine Person darstellen und in einer Kalenderwoche implementiert werden können. Die Modulbeschreibung der Blätter dieser Hierarchie muss genau genug und ausreichend sein, um das Modul implementieren zu können.

Die Modulsicht wird durch UML-Paket- und Klassendiagramme visualisiert.

Die Module werden durch ihre Schnittstellen beschrieben. Die Schnittstelle eines Moduls *M* ist die Menge aller Annahmen, die andere Module über *M* machen dürfen, bzw. jene Annahmen, die *M* über seine verwendeten Module macht (bzw. seine Umgebung, wozu auch Speicher, Laufzeit etc. gehören). Konkrete Implementierungen dieser Schnittstellen

*sind das Geheimnis des Moduls und können vom Programmierer festgelegt werden. Sie sollen hier dementsprechend nicht beschrieben werden.*

*Die Diagramme der Modulsicht sollten die zur Schnittstelle gehörenden Methoden enthalten. Die Beschreibung der einzelnen Methoden (im Sinne der Schnittstellenbeschreibung) geschieht allerdings per Javadoc im zugehörigen Quelltext. Das bedeutet, dass Ihr für alle eure Module Klassen, Interfaces und Pakete erstellt und sie mit den Methoden der Schnittstellen verseht. Natürlich noch ohne Methodenrümpfe bzw. mit minimalen Rümpfen. Dieses Vorgehen vereinfacht den Schnittstellenentwurf und stellt Konsistenz sicher.*

*Jeder Schnittstelle liegt ein Protokoll zugrunde. Das Protokoll beschreibt die Vor- und Nachbedingungen der Schnittstellenelemente. Dazu gehören die erlaubten Reihenfolgen, in denen Methoden der Schnittstelle aufgerufen werden dürfen, sowie Annahmen über Eingabeparameter und Zusicherungen über Ausgabeparameter. Das Protokoll von Modulen wird in der Modulsicht beschrieben. Dort, wo es sinnvoll ist, sollte es mit Hilfe von Zustands- oder Sequenzdiagrammen spezifiziert werden. Diese sind dann einzusetzen, wenn der Text allein kein ausreichendes Verständnis vermittelt (insbesondere bei komplexen oder nicht offensichtlichen Zusammenhängen).*

*Der Bezug zur konzeptionellen Sicht muss klar ersichtlich sein. Im Zweifel sollte er explizit erklärt werden. Auch für diese Sicht muss die Entstehung anhand der Strategien erläutert werden.*

## 5 Datensicht

*Hier wird das der Anwendung zugrundeliegende Datenmodell beschrieben. Hierzu werden neben einem erläuternden Text auch ein oder mehrere UML-Klassendiagramme verwendet. Das hier beschriebene Datenmodell wird u.a. jenes der Anforderungsspezifikation enthalten, allerdings mit implementierungsspezifischen Änderungen und Erweiterungen. Siehe die gesonderten Hinweise.*

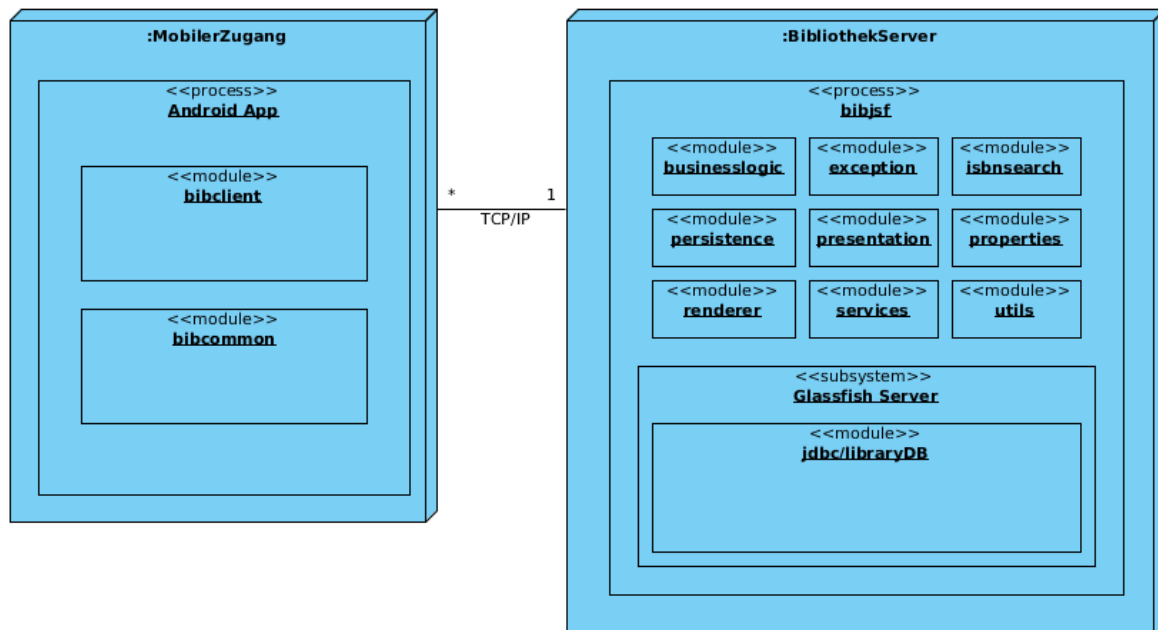
## 6 Ausführungssicht

Das folgende Diagramm [10 auf der nächsten Seite](#) zeigt das Laufzeitverhalten der Software. Auf der einen Seite haben wir den mobilen Zugang auf welchem die Android App als Prozess läuft. Die App selber verwendet die Module *bibclient* und *bibcommon*. Von diesem mobilen Zugang kann eine TCP/IP-Verbindung zu dem Bibliotheksserver aufgebaut werden. Dabei gibt es mehrere Verbindungen zu immer nur einem Server, daher die Multiplizitäten \* und 1.

Auf der anderen Seite nimmt nun der Bibliotheksserver die TCP/IP-Verbindungen an. Er ist gleichzeitig Server und Datenbankserver. Die Datenbank läuft auf dem Subsystem Glassfish Server. Der Server verwendet die Module *businesslogic*, *exception*, *isbnsearch*, *persistence*, *presentation*, *properties*, *renderer*, *services* und *util*.

Das komplette System läuft mit zwei Prozessen: einmal mit der Android App und der andere Prozess ist *bibjsf* der das Subsystem mit der Datenbank enthält. Prinzipiell gibt es unendlich viele mobile Zugänge bzw. App-Prozesse die auf einen Bibliotheksserver zugreifen.

Abbildung 10: Ausführungssicht



## 7 Zusammenhänge zwischen Anwendungsfällen und Architektur

In diesem Abschnitt sollen Sequenzdiagramme mit Beschreibung(!) für zwei bis drei von Euch ausgewählte Anwendungsfälle einen von Euch ausgewählten Anwendungsfall erstellt werden. Ein Sequenzdiagramm beschreibt den Nachrichtenverkehr zwischen allen Modulen, die an der Realisierung des Anwendungsfalles beteiligt sind. Wählt die Anwendungsfälle so, dass nach Möglichkeit alle Module Eures entworfenen Systems in mindestens einem Sequenzdiagramm vorkommen. Falls Euch das nicht gelingt, versucht möglichst viele und die wichtigsten Module abzudecken. Dazu könnt ihr Euch einen Anwendungsfall herausuchen, der möglichst viele Module der Architektur abdeckt. In SWP-2 werden wir mehrere Anwendungsfälle betrachten und eine umfangreichere Abdeckung der Architektur anstreben.

## 8 Evolution

*Beschreibt in diesem Abschnitt, welche Änderungen Ihr vornehmen müsst, wenn sich Anforderungen oder Rahmenbedingungen ändern. Insbesondere sollten hierbei die in der Anforderungsspezifikation unter „Ausblick“ bereits genannten Punkte behandelt werden.*

...